

Appendix D

Application Program Interface (API) to Distribution Services

TENA Application Program Interface - C++ Version

This appendix describes the application program interface for TENA compliant components. The version described here is a C++ language version. There are interfaces for four different component classes shown here.

The first of these is the class interface exported by Distribution Services.

The second is the interface exported by all applications assets, which is used by Distribution Services.

The third interface is that exported by Message Services and used by Distribution Services.

The fourth is the interface exported by Connection Services and used by Distribution Services.

Users should note that these are the procedural interfaces that the infrastructure and applications assets make available for use between assets and the infrastructure or between infrastructure components. There are infrastructure services that are managed via the exchange of information classes and attributes and event classes and attributes that are documented in the Information Class Catalog.

File basic_types.h

```
#ifndef BASIC_TYPES_H
```

```
#define BASIC_TYPES_H
```

```
//
```

```
// This file defines the basic types which the TENA provides
```

```
// to all code modules.
```

```
//
```

```
//
```

```
// TENA Application Program Interface C++ version
```

```
//
```

```
//  
// Basic arithmetic types  
  
//  
  
typedef int Int;  
typedef short Short_Int;  
typedef unsigned int Unsigned_Int;  
typedef unsigned short int Unsigned_Short_Int;  
typedef char Char;  
typedef float Float;  
typedef double Double;  
typedef char Octet;  
  
//  
// Fundamental constants  
  
//  
  
#ifndef NULL  
#define NULL (0)  
#endif
```

File distrib_serv_except.h

```
#ifndef DISTRIB_SERV_EXCEPT_H  
#define DISTRIB_SERV_EXCEPT_H  
  
//  
// TENA Application Program Interface C++ Version  
  
//  
  
// Basic class for exceptions supported by distribution services.
```

```
//  
class Exception {  
public:  
    const char    *_cause;           //user provided exception trigger  
    const char    *_name;           //id of the exception  
// constructors  
Exception(const char *cause) { _cause = cause; }  
Exception(const Exception &copy)  
{  
    _cause = copy._cause;  
    _name = copy._name;  
}  
// Destructor  
virtual ~Exception() {}  
// Operators  
Exception& operator= (const Exception&);  
};  
//  
// Macro for defining exceptions as subclasses of Exception.  
//  
#define TENAEXCEPTION(E) \  
class E : public Exception { \  
public: \  
    E(const char *cause) : Exception(cause) {} \  
    E(const Exception &copy) : Exception(copy) {} \  
};
```

```
#endif

file distrib_serv_types.h

#ifndef DISTRIBUT_SERV_TYPES_H
#define DISTRIBUT_SERV_TYPES_H
#include "basic_types.h"
#include "distrib_serv_except.h"
//
// TENA Application Program Interface C++ Version
//
//
// Exceptions thrown by distribution services components.
//
TENAEXCEPTION(AlreadyPublished)
TENAEXCEPTION(DestinationNotSubscribed)
TENAEXCEPTION(FilterNotActive)
TENAEXCEPTION(InvalidAssetID)
TENAEXCEPTION(InvalidAttribute)
TENAEXCEPTION(InvalidClass)
TENAEXCEPTION(InvalidFilterType)
TENAEXCEPTION(InvalidGroupID)
TENAEXCEPTION(InvalidInstanceID)
TENAEXCEPTION(InvalidParameter)
TENAEXCEPTION(NotEventClass)
TENAEXCEPTION(NotPublished)
TENAEXCEPTION(NotSubscribed)
TENAEXCEPTION>PasswordFailure)
```

```
TENAEXCEPTION(QueueFull)
TENAEXCEPTION(StillRegistered)
TENAEXCEPTION(UnknownAsset)
TENAEXCEPTION(UnknownChannel)
TENAEXCEPTION(UnknownDestination)
TENAEXCEPTION(UnknownInstance)

//=====
//
// Constants for configuration management.
//
#define      MAX_ATTRIB_PER_CLASS      100
#define      MAX_ATTRIBUTE_VALUE_SIZE 64
#define      MAX_FILTER_PARAM_VALUE_SIZE 8
#define      TENA_VERSION           "0.1R1"

//=====
//
// Basic type definitions for parameter elements.
//
typedef      Unsigned_IntAttribute_Descriptor;
typedef      Unsigned_Int      Class_Descriptor;
typedef      Unsigned_Int      QOS_Descriptor;
typedef      Unsigned_Int      Instance_Descriptor;
typedef      Unsigned_Int      Filter_Parameter_Descriptor;
typedef      Unsigned_Int      Channel_Identifier;
typedef      Unsigned_IntAsset_Identifier;
```

```
typedef      Unsigned_IntAccess_Group_Identifier;
typedef      Char          *Access_Group_Password;
enum Boolean {FALSE, TRUE};
enum Filter_Descriptor {TIME_SAMPLED, ONE_OF_N_SAMPLED, REGION};

//  

// Definitions for messages handled by Distribution Services.  

//  

//  

// Structure defining the time representation for message timestamps. This is  

// always Greenwich time where Sec is the number of seconds since January 1, 1970  

// and Microsec is the number of microseconds.  

//  

struct Date_Time {  

    Unsigned_IntSec;  

    Unsigned_IntMicrosec;  

};  

//  

// Message header.  

//  

// SourceID is the identifier of the sender.  

// DestinationID is the identifier of the end destination component. If NULL the  

// message is broadcast to all possible receivers.  

// DataSize is the number of octets in the message body.  

// TimeSent is the time at which the message was dispatched by the sender.  

//
```

```

struct DS_Message_Header {
    Asset_Identifier      SourceID;
    Asset_Identifier      DestinationID;
    Int                  DataSize;
    Date_Time            TimeSent;
};

// Message packet.

// Hdr is the header for this message.

// Data is a pointer to the message body.

// =====

// Class definition for a descriptor set.

class Descriptor_Set {
private:
    Int          _size;        //current size of the set
    Attribute_Descriptor_Set[MAX_ATTRIB_PER_CLASS];
public:
    // constructors

```

```
Descriptor_Set();
// set manipulation

void AddDescriptor(Attribute_Descriptor a);
Attribute_Descriptor GetDescriptor();
void ClearSet();
Boolean Empty();
};

//


// Class definition for an attribute descriptor, value pair set.

//
class Attribute_Value_Set {

private:
struct avs {
    Attribute_Descriptor d;
    Char           val[MAX_ATTRIBUTE_VALUE_SIZE];
    Int            length;
};

    Int      _current;
    Int      _size;
    avs   *_set;

public:
// constructors

    Attribute_Value_Set(Int size) ;
// destructor

    ~Attribute_Value_Set();

// set manipulation
```

```
Int Size() const;

void AddDescriptorValue(
    Attribute_Descriptor attrib,
    Char           *value,
    Int            length) ;

void GetDescriptorValue(
    Int            index,
    Attribute_Descriptor *descrip,
    Char           *value,
    Int            *length);

void Clear();

};

//  

// Class definition for a quality of service descriptor, value pair set.  

//  

class QOS_Value_Set {  

private:  

    struct avs {  

        QOS_Descriptor   d;  

        Char            val[MAX_ATTRIBUTE_VALUE_SIZE];  

        Int             length;  

    };  

    Int      _current;  

    Int      _size;  

    avs    *_set;  

public:
```

```

// constructors

QOS_Value_Set(Int size) ;

// destructor

~QOS_Value_Set();

// set manipulation

Int Size() const;

void AddDescriptorValue(
    QOS_Descriptor      attrib,
    Char                *value,
    Int                 length) ;

void GetDescriptorValue(
    Int                  index,
    QOS_Descriptor      *descrip,
    Char                *value,
    Int                 *length);

void Clear();

};

//


// Class definition for a filter parameter, value pair set.

//

class Filter_Value_Set {

private:

struct fvs {

Filter_Parameter_Descriptor      d;
Char                            val[MAX_FILTER_PARAM_VALUE_SIZE];
Int                             length;

```

```

};

Int    _current;
Int    _size;
fvs   *_set;

public:
// constructors
Filter_Value_Set(Int size) ;
// destructor
~Filter_Value_Set();
// set manipulation
Int Size() const;
void AddParameterValue(
Filter_Parameter_Descriptor      param,
Char                *value,
Int                 length);
void GetParameterValue(
Int                  index,
Filter_Parameter_Descriptor    *descrip,
Char                *value,
Int                 *length);
void Clear();
};

// 
// Parameter definitions for service methods.
//
typedef      Descriptor_Set      Attribute_Descriptor_Set;

```

```
typedef      void (*Update_Handle)(  
                                const Asset_Identifier,  
                                const Instance_Descriptor,  
                                Attribute_Value_Set&);  
  
//================================================================  
  
#endif  
  
file distrib_serv.h  
  
#ifndef DISTRIB_SERV_H  
#define DISTRIB_SERV_H  
  
#include "basic_types.h"  
#include "distrib_serv_except.h"  
#include "distrib_serv_types.h"  
  
//  
  
// TENA Application Program Interface C++ Version  
  
//  
  
//  
  
// This function is defined to provide an interface for the update  
// service offered by Distribution Services. It is located outside the  
// class definition to allow any appropriate function address to be  
// used as a method handle, including those outside of Distribution Services  
// and also outside the infrastructure.  
  
//  
  
// Destination is the identifier of a destination asset. This is NULL  
// when the update is delivered to all subscribers and non NULL when
```

```
// the update is issued in response to a RequestUpdate in which case
// this is set to the requester's identifier.

// InstanceID is the instance of the class which the update pertains to.

// AttributeValueList is a container which holds attribute descriptor,
// attribute value pairs.

//

void DSUpdate(
    const Asset_Identifier           Destination,
    const Instance_Descriptor InstanceID,
    Attribute_Value_Set      &AttributeValueList)
throw (
    NotPublished,
    InvalidAttribute,
    InvalidInstanceID);

//


// This class definition encapsulates Distribution Services and
// makes available a number of services for system components to
// communicate information about their state.

//

class DistServ {
public:
    #include "distrib_serv_api.h"
};

#endif
```

```
file distrib_serv_api.h

#ifndef DISTRIB_SERV_API_H
#define DISTRIB_SERV_API_H

//

// TENA Application Program Interface C++ Version
//


////////////////////

// Publish //

////////////////////

//

// Service to publish an information class.

//

// AssetID is the identifier of the asset publishing the class.

// ClassD is the descriptor of the class being published.

// AttributeList is a container which holds the descriptors of the

// attributes which are being published for the indicated class.

// QualityOfService is a container which holds pairs of quality-of-

// service descriptors and their value. This defaults to NULL. A

// NULL value implies the implementation defaults for quality-of-

// service will be used.

// GroupID is an identifier of an information group which the publisher

// of the class is assigning any class instances registered by this

// publisher to. The default is NULL which implies unlimited access.

// GroupPassword is the password for the group indicated. Subscribers
```

```
// must provide an identical password to obtain access to information
// provided by this publisher about all instances it registers of this
// class.

//
// The function returns a function pointer or handle to be used for
// delivery of attribute updates.

//
Update_Handle
Publish(
    const Asset_Identifier           AssetID,
    const Class_Descriptor          ClassD,
    Attribute_Descriptor_Set        AttributeList,
    QOS_Value_Set                   QualityOfService =0,
    const Access_Group_Identifier   GroupID =0,
    const Access_Group_Password     GroupPassword =0)
throw (
    AlreadyPublished,
    InvalidAttribute,
    InvalidClass,
    InvalidGroupID);

/////////////////////
// Unpublish //
/////////////////////
//
```

```
// Service to use to cease publishing a class.  
//  
// AssetID is the identifier of the asset requesting the service.  
// ClassD is the descriptor of the class which it wishes to cease publishing.  
//  
void  
Unpublish(  
    const Asset_Identifier      AssetID,  
    const Class_Descriptor     ClassD)  
throw (  
    NotPublished,  
    InvalidAssetID,  
    InvalidClass,  
    StillRegistered);
```

```
///////////  
// RegisterInstance //  
///////////  
//  
// Service used to register an instance of a class which is being  
// published by the service requester.  
//  
// AssetID is the identifier of the service requester.  
// ClassD is the descriptor of the class which is being registered.  
//
```

```
// The function returns the instance identifier.  
//  
Instance_Descriptor  
RegisterInstance(  
    const Asset_Identifier      AssetID,  
    const Class_Descriptor     ClassD)  
throw (  
    NotPublished,  
    InvalidAssetID,  
    InvalidClass);  
  
///////////  
// SendEvent //  
///////////  
//  
// Service to send an event. This implies a temporary instantiation of the  
// class, update of attribute values, and removal of the temporary instance.  
// This is applied to event classes for which instances are not registered.  
//  
// AssetID is the identifier of the asset requesting service.  
// ClassD is the descriptor of the class of the event is being sent.  
// AttributeValueList is a container which holds pairs of attribute  
// descriptors and their value.
```

```
// Destination is the asset identifier of the asset to which the event is
// to be sent, if delivery to a specific asset is required. If NULL the
// event will be sent to all subscribers.

//
void

SendEvent(
    const Asset_Identifier      AssetID,
    const Class_Descriptor     ClassD,
    Attribute_Value_SetAttributeValueTypeList,
    const Asset_Identifier      Destination)
throw (
    NotPublished,
    InvalidAssetID,
    InvalidClass,
    InvalidAttribute,
    NotEventClass,
    DestinationNotSubscribed);
```

```
///////////
// RemoveInstance //
///////////
//
// Service used to remove an instance of a class from the system. This
// announces the publishers intent to not send any further updates of
```

```
// attribute values for the instance.  
//  
// AssetID is the identifier of the asset requesting service.  
// ClassD is the descriptor of the class of the instance.  
// InstanceID is the identifier of the instance to be removed.  
//  
void  
RemoveInstance(  
    const Asset_Identifier          AssetID,  
    const Class_Descriptor         ClassD,  
    const Instance_Descriptor     InstanceID)  
throw(  
    InvalidInstanceID,  
    InvalidAssetID,  
    UnknownInstance);  
  
///////////  
// Subscribe //  
///////////  
//  
// Service to subscribe to an information class.  
//  
// AssetID is the identifier of the asset requesting the service.  
// ClassD is the descriptor of the class which the requester is subscribing to.
```

```
// AttributeList is a container holding a list of the attribute descriptors
// that the subscriber desired be delivered.

// QualityOfService is a container which holds pairs of quality-of-
// service descriptors and their value. This defaults to NULL. A
// NULL value implies the implementation defaults for quality-of-
// service will be used.

// GroupID is an identifier of an information group which the publisher
// of the class has assigned any class instances registered by the
// publisher to. The default is NULL which implies unlimited access.

// The subscriber will get only those class instances assigned to the group.

// GroupPassword is the password for the group indicated. Subscribers
// must provide an identical password to obtain access to information
// provided by the publishers about all instances they register of this
// class.

//
void

Subscribe(
    const Asset_Identifier      AssetID,
    const Class_Descriptor     ClassD,
    Attribute_Descriptor_Set   AttributeList,
    QOS_Value_Set               QualityOfService =0,
    const Access_Group_Identifier GroupID =0,
    const Access_Group_Password GroupPassword =0)
throw (
    InvalidAttribute,
    InvalidClass,
    InvalidAssetID,
```

```
InvalidGroupID,  
PasswordFailure);  
  
//////////  
// Unsubscribe //  
//////////  
//  
// Service to announce that a subscriber is no longer interested in  
// subscribing to the indicated class.  
//  
// AssetID is the identifier of the asset publishing the class.  
// ClassD is the descriptor of the class being published.  
//  
void  
Unsubscribe(  
    const Asset_Identifier      AssetID,  
    const Class_Descriptor     ClassD)  
throw (  
    InvalidClass,  
    InvalidAssetID,  
    NotSubscribed);
```

```
//////////  
// ActivateFilter //  
//////////  
//  
// Service to activate a filter in an information channel.  
//  
// AssetID is the identifier of the asset requesting the service.  
// ClassD is the descriptor of the class the filter is to be applied to.  
// FilterType is the type of filter to activate.  
// FilterParameterSet is a container holding pairs of filter parameter  
// descriptors and their value.  
//  
void  
ActivateFilter(  
    const Asset_Identifier      AssetID,  
    const Class_Descriptor     ClassD,  
    const Filter_Descriptor    FilterType,  
    Filter_Value_Set           FilterParameterSet)  
throw (  
    InvalidAssetID,  
    InvalidClass,  
    InvalidFilterType,  
    NotSubscribed,  
    InvalidParameter);
```

```
//////////  
// ModifyFilter //  
//////////  
//  
// Service to modify a filter's parameters.  
//  
// AssetID is the identifier of the asset requesting the service.  
// ClassD is the descriptor of the class the filter is applied to.  
// FilterType is the type of filter to modify.  
// FilterParameterSet is a container holding pairs of filter parameter  
// descriptors and their value.  
//  
void  
ModifyFilter(  
    const Asset_Identifier      AssetID,  
    const Class_Descriptor     ClassD,  
    const Filter_Descriptor    FilterType,  
    Filter_Value_Set   FilterParameterSet)  
throw (  
    InvalidAssetID,  
    InvalidClass,  
    InvalidFilterType,  
    NotSubscribed,
```

```
    InvalidParameter,  
    FilterNotActive);  
  
//////////  
// DeactivateFilter //  
//////////  
//  
// Service to deactivate a filter in an information channel.  
//  
// AssetID is the identifier of the asset requesting the service.  
// ClassD is the descriptor of the class the filter is applied to.  
// FilterType is the type of filter to deactivate.  
//  
void  
DeactivateFilter(  
    const Asset_Identifier      AssetID,  
    const Class_Descriptor     ClassD,  
    const Filter_Descriptor    FilterType)  
throw (  
    InvalidAssetID,  
    InvalidClass,  
    InvalidFilterType,  
    NotSubscribed,
```

```
FilterNotActive);  
  
//////////  
// RequestUpdate //  
//////////  
//  
// Service to query a publisher or publishers for the current value of  
// indicated attributes for the class or classs instances provided.  
//  
// AssetID is the identifier of the asset requesting the service.  
// ClassD is the descriptor of the class being requested.  
// AttributeList is a container which holds the descriptors of the  
// attributes which are being requested for the indicated class.  
// InstanceID is the identifier for the instance requested. If NULL  
// then updates will be generated for all instances of the class  
// registered by all publishers.  
//  
void  
RequestUpdate(  
    const Asset_Identifier      AssetID,  
    const Class_Descriptor     ClassD,  
    Attribute_Descriptor_Set  AttributeList,  
    const Instance_Descriptor InstanceID =0)
```

```
throw (
    InvalidInstanceID,
    UnknownInstance,
    InvalidAttribute,
    InvalidClass,
    InvalidAssetID,
    NotSubscribed);
```

```
///////////
// RegisterChannel //
///////////
//
// Service used by Connection Services to advise Distribution Services that it
// has obtained access to a communications channel which supports the quality-of-
// service description provided.
```

```
//
// ChannelID is the identifier of the channel which will be used by both
// Distribution Services and Connection Services to refer to the described
// channel.
```

```
// CircuitDescription is a container which holds a description of the channel
// which is supported. This consists of pairs of quality-of-service parameter
// descriptors and their value.
```

```
//
void
```

```
RegisterChannel(  
    Channel_Identifier  ChannelID,  
    QOS_Value_Set        CircuitDescription,  
    Update_Handle        Method)  
throw (  
    InvalidParameter);
```

```
//////////  
// UnregisterChannel //  
//////////  
//  
// Service used by Connection Services to advise Distribution Services that it  
// no longer supports the channel indicated.  
//  
// ChannelID is the identifier of the channel.  
//  
void  
UnregisterChannel(  
    Channel_Identifier  ChannelID)  
throw (  
    UnknownChannel);
```

```
//////////  
// TerminateAsset //  
//////////  
//  
// Service to clean up traces of an asset if it terminates without being able to  
// unpublish and unsubscribe.  
//  
// AssetID is the identifier of the asset requesting the service. This asset must  
// be authorized to terminate the target asset.  
// TargetID is the asset identifier of the asset which is to be terminated.  
//  
void  
TerminateAsset(  
    const Asset_Identifier      AssetID,  
    const Asset_Identifier      TargetID)  
throw (  
    InvalidAssetID,  
    UnknownAsset);
```

```
//////////
```

```
// Receive //
```

```
///////////  
//  
// This method is used as a general purpose message delivery method. It is used by  
// lower level services in the communications channels to deliver incoming  
// messages when operating in an interrupt driven mode.  
//  
// Message is a pointer to the message packet.  
// ChannelID is the identifier of the channel the message was received on.  
//  
void  
Receive(  
    DS_Message_Pkt *Message,  
    const Channel_Identifier ChannelID)  
throw (  
    UnknownDestination,  
    UnknownChannel,  
    QueueFull);  
  
#endif
```

```
file asset.h  
#ifndef ASSET_H  
#define ASSET_H  
#include "basic_types.h"  
#include "distrib_serv_except.h"  
#include "distrib_serv_types.h"
```

```
//  
// TENA Application Program Interface C++ Version  
//  
//  
// This class definition encapsulates the interface between any system  
// component with an asset identifier and Distribution Services. It  
// makes available a number of methods which enable Distribution Services  
// to deliver data and control information to the component.  
//  
//  
class Asset {  
public:  
#include "asset_api.h"  
};  
#endif  
  
file asset_api.h  
#ifndef ASSET_API_H  
#define ASSET_API_H  
  
//  
// TENA Application Program Interface C++ Version  
//
```

```
//////////  
// RemoveInstance //  
//////////  
//  
// Service used to advise an asset that an instance has been removed.  
// announces the publishers intent to not send any further updates of  
// attribute values for the instance.  
//  
// AssetID is the identifier of the asset requesting service.  
// ClassD is the descriptor of the class of the instance.  
// InstanceID is the identifier of the instance to be removed.  
//  
void  
RemoveInstance(  
    const Class_Descriptor    ClassD,  
    const Instance_Descriptor InstanceID)  
throw(  
    InvalidInstanceID,  
    UnknownInstance);
```

```
//////////  
// RequestUpdate //  
//////////
```

```
//  
// Service to query a publisher or publishers for the current value of  
// indicated attributes for the class or classs instances provided.  
//  
// AssetID is the identifier of the asset requesting the service.  
// ClassD is the descriptor of the class being requested.  
// AttributeList is a container which holds the descriptors of the  
// attributes which are being requested for the indicated class.  
// InstanceID is the identifier for the instance requested. If NULL  
// then updates will be generated for all instances of the class  
// registered by all publishers.  
//  
void  
RequestUpdate(  
    const Asset_Identifier      AssetID,  
    const Class_Descriptor     ClassD,  
    Attribute_Descriptor_Set   AttributeList,  
    const Instance_Descriptor  InstanceID =0)  
throw (  
    InvalidInstanceID,  
    UnknownInstance,  
    InvalidAttribute,  
    InvalidClass,  
    InvalidAssetID,  
    NotSubscribed);
```

```
//////////  
// SendEvent //  
//////////  
//  
// This method is used by Distribution Services to send an event to the asset.  
//  
// AssetID is the identifier of the asset originating the event.  
// ClassD is the descriptor of the class of the event.  
// AttributeValueList is a container which holds pairs of attribute  
// descriptors and their value.  
//  
void  
SendEvent(  
    const Asset_Identifier      AssetID,  
    const Class_Descriptor     ClassD,  
    Attribute_Value_SetAttributeValueType  
throw (  
    InvalidAssetID,  
    InvalidClass,  
    InvalidAttribute,  
    NotSubscribed);
```

```
///////////  
// Update //  
///////////  
//  
// This method is used by Distribution Services to deliver updates  
// of attribute values for classes to which the asset is subscribed.  
//  
// ClassD is the descriptor of the class of the instance.  
// InstanceID is the instance of the class which the update pertains to.  
// AttributeValueList is a container which holds attribute descriptor,  
// attribute value pairs.  
//  
void  
Update(  
    const Class_Descriptor    ClassD,  
    const Instance_Descriptor InstanceID,  
    Attribute_Value_Set       &AttributeValueList)  
throw (  
    NotSubscribed,  
    InvalidAttribute,  
    InvalidInstanceID);  
  
///////////
```

```
// DiscoverInstance //
///////////
//
// This method is used by Distribution Services to advise an asset
// that is subscribed to the indicated class that an instance of that
// class has been registered.
//
// ClassD is the descriptor of the class of the instance.
// InstanceID is the identifier of the instance.
//
void
DiscoverInstance(
    const Class_Descriptor    ClassD,
    const Instance_Descriptor InstanceID)
throw (
    NotSubscribed,
    InvalidClass,
    InvalidInstanceID);
```

```
///////////
// StopUpdating //
///////////
//
// This method is used by Distribution Services to advise an asset
```

```
// publishing a class to stop issuing updates for that class.  
//  
// ClassD is the descriptor of the class the request applies to.  
//  
void  
StopUpdating(  
const Class_Descriptor    ClassD)  
throw (  
NotPublished,  
InvalidClass);
```

```
///////////  
// ResumeUpdating //  
///////////  
//  
// This method is used by Distribution Services to advise an asset  
// publishing a class to resume issuing updates for instances of that class.  
//  
// ClassD is the descriptor of the class the request applies to.  
//  
void  
ResumeUpdating(  
const Class_Descriptor    ClassD)  
throw (
```

```
NotPublished,  
InvalidClass);  
  
#endif  
  
  
  
file message_serv.h  
#ifndef MESSAGE_SERV_H  
#define MESSAGE_SERV_H  
  
#include "basic_types.h"  
#include "distrib_serv_except.h"  
#include "distrib_serv_types.h"  
  
//  
// TENA Application Program Interface C++ Version  
//  
  
//  
// This class definition provides the part of the Message Services -  
// Distribution Services interface which Distribution Services uses  
// to deliver data and control information to Message Services.  
//  
class Message_Services {  
public:  
#include "message_serv_api.h"  
};  
#endif
```

```
file message_serv_api.h

#ifndef MESSAGE_SERV_API_H
#define MESSAGE_SERV_API_H


//


// TENA Application Program Interface C++ Version
//


///////////////////////


// RegisterCallback //
///////////////////////


//


// Service used to advise that the requester wishes to receive an
// interrupt when a message has been received on the indicated protocol.
//


// Protocol is the identifier of the protocol channel the service applies to
// Method is the handle of the interrupt service routine
//


void

RegisterCallback(
    const Channel_Identifier    Protocol,
    const Receive_Handle        Method)
throw(
    InvalidHandle,
    UnknownChannel);
```

```
///////////  
// CancelCallback //  
///////////  
//  
// Service used to advise that the requester wishes to cancel a  
// callback previously registered.  
//  
// Protocol is the identifier of the protocol channel the service applies to  
//  
void  
CancelCallback(  
    const Channel_Identifier    Protocol)  
throw(  
    NotRegistered,  
    UnknownChannel);
```

```
///////////  
// Send //  
///////////  
//  
// This method is used to dispatch a message.
```

```
//  
// Data is a pointer to the message data.  
// Protocol is the identifier of the protocol channel to use.  
//  
void  
Send(  
    const DS_Message_Pkt *Data,  
    const Channel_Identifier Protocol)  
throw (  
    InvalidAssetID,  
    UnknownChannel,  
    InvalidAddress,  
    UnableToSend);
```

```
///////////  
// Receive //  
///////////  
//  
// This method is used to retrieve the next message on the indicated  
// protocol queue.  
//  
// Protocol is the identifier of the protocol channel to use.  
// The function returns a pointer to the message. The pointer  
// is set to NULL if there are no messages queued.
```

```
//  
DS_Message_Pkt *  
Receive(  
const Channel_Identifier Protocol)  
throw (  
UnknownChannel);  
#endif
```

```
file conx_serv.h  
#ifndef CONX_SERV_H  
#define CONX_SERV_H  
#include "basic_types.h"  
#include "distrib_serv_except.h"  
#include "distrib_serv_types.h"  
#include "message_serv.h"  
  
//  
// TENA Application Program Interface C++ Version  
  
//  
  
//  
// This class definition provides the part of the Connection Services -  
// Distribution Services interface which Distribution Services uses  
// to deliver data and control information to the Connection Services.  
  
//  
class Connection_Services : public Message_Services {
```

};

#endif